Predicting and Understanding Genomic Sequences Using Neural Networks

By Sarah Cross

Table of Contents

Background(Shorter Version)

The virion, or the complete form of a virus outside of the host cell, is enveloped, spherical, and contains one copy of the positive-sense RNA genome. A virus spreads by injecting its genome into the host cell, and this genome is then translated into another viral protein, RNA, or ribosomal nucleic acid. This then forms a new virus, which leaves the cell to go and infect other cells. Viruses depend on the host ribosome to make its protein, and the host ribosome only read mRNA, so viruses are forced to translate their genome into mRNA. From there the ribosome then creates the viral proteins. Positive RNA is when the genome is translated directly into mRNA, meaning that the ribosome can directly read the genome without the virus having to translate the viral protein or DNA into mRNA. Viruses must learn to adapt ecological niches or they become extinct.

Potential cancer treatments have been suggested by modulating transcription factors and “emphasiz[ing] agents with established clinical efficacy or with promising effects in preclinical models.”

Deep Neural Networks have proven to be incredibly accurate at obtaining sites of transcription factor(TF) binding(TFBS). The Deep Motif Dashboard, for example, provides a suite of visualization strategies to extract motifs or sequence patterns from deep neural networks for TFBS classification without the use of external sources such as protein maps.

Transcription factors are regulatory proteins that bind to DNA(turn genes “on” and “off”). Given an input DNA sequence, DeMo classifies whether or not there’s a binding site for a TF. Given a TF of interest and a dataset made of samples of positive and negative TFBS sequences, the model tests three DL architectures to classify the sequences: CNN, RNN, and CNN-RNN structures. Secondly, DeMo attempts to understand why they perform the way they do by measuring nucleotide importance with saliency maps, measuring critical sequences positions for the classifier using temporal output scores, and generating class-specific motif patterns with class optimization.

Chromatin immunoprecipitation(ChIP-seq) technologies, or the precipitation of a protein antigen used to isolate and concentrate a protein from a sample containing thousands of proteins, makes finding site locations available for hundreds of different TFs. Unfortunately, ChIP-seq experiments are slow, expensive, and they can’t find patterns common across binding sites(although they can find the binding sites).

Using a deep neural network, three different architectures were tested: CNN, RNN, and a combination of the two, CNN-RNN. The raw nucleotide base characters are used as input to output an output vector of a fixed dimension, which is linearly fed to a softmax function. The final output returns a binary classification task of size 1x1 returning whether the input is a positive or negative binding site. The models use the stochastic gradient algorithm Adam with a mini-batch size of 256 sequences and a dropout

regularization method. DeMo attempts to understand which parts of the DNA sequence are the most influential for classification by using saliency maps. Given a sequence X_0 of length $|X_0|$ and class $c \in C$, a DNN model provides a score function of $S_c(X_0)$. Because it's difficult to directly see the influence of each nucleotide on the complex, highly non-linear score function, given X, $S_c(X)$ can be approximated by a linear function by computing the first-order Taylor expansion(a representation of a function as an infinite sum of terms that are calculated from the values of the function's derivatives at a single point).

With bacterial genomes, codons are very easy to identify and find. Unfortunately there's more space in between real genes in eukaryotic genes(62% of human genome is intergenic), there are introns which interrupt the sequencing of DNA, codon bias(not all codons are used equally frequently), exon-intron boundaries, and many other problems.

Biologists in recent years have also been able to identify transcription factors(regulatory proteins that bind to a particular sequence), DNase I hypersensitive sites(sites sensitive to cleavage by the DNase I enzyme), and histone marks(chemical modifications to histone proteins). Given no additional outside information, the RNN attempts to predict whether a given feature will be present given only the sequence at the nucleotide-level.

Other models have been published in the past. DeepSEA, for example, is a deep neural network consisting of a mixture of convolutional neural networks and

pooling layers which worked on a fixed length sequence, along with a sigmoid output layer to compute the probability of seeing a particular genome feature. DeepBind used a deep CNN with 4 stages of convolution, rectification, pooling, and nonlinear activation functions.

Basset, published a few months later, used a deep CNN architecture to learn the functional activity of genomics sequences.

DanQ, published in December 2015, incorporated a convolutional layer and a bidirectional long short term memory(LSTM) recurrent neural network(RNN) on top of a max pooling layer.

All of the following neural networks utilized one-hot encoding of the input sequence into the convolution layer. One-hot encoding expands integer inputs into arrays of a fixed length; for example, given that the nucleotide bases A, T, C, and G are given integer values, each letter in the sequence of DNA of length *L* would be transformed into a 4 x *L* matrix. Each tool used a convolutional layer first, which requires fixed sizes. By introducing several sets of convolutional layers, more parameters are introduced, thus a first approach should avoid using convolutional layers.

The following loss function was used by the researchers:

$$PP(y,\hat{y}) = exp\{-\frac{1}{n-1}\sum_{t=1}^{n-1}\sum_{i=1}^{|V|} y_i^{(t)} log\hat{y}_i^{(t)}\}$$

For which *n* is the number of training samples(about the length of the genome), |V| is the size of the prediction set(4 for the 4 nucleotide bases),

$\hat{y}_i^{(t)}$ is the predicted probability of the predicted word at time t being word i, and $y_i^{(t)}$ is a one-hot vector of the real word at time t.

For character-level processing, a bidirectional GRU is used because a genetic sequence can be regulated by both the beginning and end sequences. The outputs then go through a softmax layer; for each K task, a prediction of 0 or 1 is made which indicates whether or not a particular feature should be observed for the input sequence. The overall loss is the geometric average of the perplexity across all K tasks. For the multitask prediction problem, |V| equals 2, the number of classes for each class.

The researchers used a random subset of the dataset used in the DanQ and DeepSEA papers. The dataset was collected from experiments in the ENCODE and Roadmap Epigenomics data releases. They randomly chose 80,000 sequence-label pairs for the training set, 8000 for the testing set, and 2000 for the validation set. Each sequence has a length of 1000.

At the time, training an RNN on all 1000 characters in each example proved to be particularly difficult due to computational limitations. As a result, the RNNs below were trained on the middle 100 characters of each sequence. The baseline model was trained on the same set of truncated sequences.

The first two datasets are used as testing in order to filter out any model not capable of performing well on these genomes.

For the *E. coli* and *P. falciparum* genomes, the researchers used a dropout keep rate of 1, a batch size of 50, and 1 epoch. They tested: “GRU, LSTM, and

simple RNN architectures, learning rates of 0.0001 and 0.0008, sequence lengths of 50, 100, 500, and 1000, and 2 and 3 layers." All combinations of the architectures returned perplexities of 3.679 for *E. coli* and 2.938 for *P. falciparum*. Although the perplexity was low, training for more than 1 epoch showed a reduction in the perplexity; however, in this experiment the number of epochs was reduced in order to test the hyperparameters and their influence on the perplexity.

Using a learning rate of 0.0008 was much faster than using a learning rate of 0.0001, although this is not surprising considering that smaller perplexities are possible and thus increasing the learning rate in this instance would not lead to diverging from the local minima.

Using 2 or 3 layers did not significantly influence the model's ability to return smaller perplexities, and, although genomes often involve long-range interactions, sequences of 50 or 100-length sequences performed better than 500 or 1000-length sequences. Overall, character-level genome prediction results proved that a 2-layer network with GRUs would be most appropriate for modeling a genomic sequence and that longer input sequences do not necessarily mean lower perplexities.

Logistic regression was used with default parameters excluding the regularization constant of the l_2 regularization term set to 10^{-6}, and, instead of using raw error rate, the researchers looked at the F1 score defined as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

For which precision is the proportion of correct positive predictions and recall is the proportion of positive examples that received a positive prediction. A perfect score would be 1(meaning the model has reached perfect precision and recall) and its worst at 0.

Ultimately, the final model used a learning rate of 0.001, 2 hidden layers, an embedding size of 2(because there were only 4 unique characters), a dropout keep probability of 0.95, hidden layer sizes of 128, a bidirectional RNN GRU with tanh activation functions, an input sequence of length 100, an Adam optimizer, and hidden layer with a size of 128.

The model, after training for 61 epochs on 80,000 training examples for 3 days on the NVIDIA GRID K520 GPU, plateaued with an average validation loss trapped at a perplexity of 1.087.

In comparison, the RNN beat the logistic regression model in 94.6% of the tasks, giving an average F1 score of 0.135 in comparison to the logistic regression model's average score of 0.044.

In conclusion, this project constructed deep bidirectional RNNs capable of predicting and capturing complex patterns in 100-character sequences of the human genome. The researchers concluded that more improvements could be made to their model:

1) Creating more data, for example, extending the 100-character sequences to 200-character sequences.

2) Modifying the hyperparameters further.

3) Designing an RNN cell suited for the long-range information associated with genomic sequences such as the clockwork RNN architecture.

4) Finding a distributed representation of genomic “words” and initializing a word-based RNN model.

Sequence prediction and classification are problems Recurrent Neural Networks(RNNs) are both designed and used for. In theory RNNs have the ability to cope with temporal dependencies; however, when long-term memory is required, they are difficult to train correctly.

Background(Long Version)

The virion, or the complete form of a virus outside of the host cell, is enveloped, spherical, and contains one copy of the positive-sense RNA genome. A virus spreads by injecting its genome into the host cell, and this genome is then translated into another viral protein, RNA, or ribosomal nucleic acid. This then forms a new virus, which leaves the cell to go and infect other cells. Viruses depend on the host ribosome to make its protein, and the host ribosome only read mRNA, so viruses are forced to translate their genome into mRNA. From there the ribosome then creates the viral proteins. Positive RNA is when the genome is translated directly into mRNA, meaning that the ribosome can directly read the genome without the virus having to translate the viral protein or DNA into mRNA. Viruses must learn to adapt ecological niches or they become extinct. Scientists have found that several bird species are capable of generating extremely high viremias, which, as a result, causes mosquitoes to become infected as well. Viremia is a medical condition in which viruses enter the bloodstream, giving them access to the entire body. Because mosquitoes bite other animals in order to feed off of their blood, the virus then spreads to them. Viremia in Crows can spread to more than 10^{10} mL of blood, and the mortality is almost uniform. There are many different lineages of WNV, but only the first two lineages are universally accepted.

Potential cancer treatments have been suggested by modulating transcription factors and “emphasiz[ing] agents with established clinical efficacy or with promising effects in preclinical models.” Anand S. Bhagwat and Christopher R. Vakoc, from Cold Spring Harbor Laboratory, have been working on pharmacologically elevating the function of specific TFs related to tumor suppression pathways.

Deep Neural Networks have proven to be incredibly accurate at obtaining sites of transcription factor(TF) binding(TFBS).

The Deep Motif Dashboard provides a suite of visualization strategies to extract motifs or sequence patterns from deep neural networks for TFBS classification without the use of external sources such as protein maps. Finding a test sequence saliency map using first-order derivatives to find the importance of each nucleotide in TFBS classification.

Understanding genomic sequences is important because of its high correlation of genes with diseases and drugs. By understanding genomic sequences, we can better understand how mutations such as single nucleotide polymorphisms or methylation can change phenotypes and allow scientists to design more effective treatments for genetic illnesses.

Transcription factors are regulatory proteins that bind to DNA(turn genes “on” and “off”). Given an input DNA sequence, DeMo classifies whether or not there’s a binding site for a TF. Given a TF of interest and a dataset made of

samples of positive and negative TFBS sequences, the model tests three DL architectures to classify the sequences:

- CNN
- RNN
- CNN-RNN

Secondly, DeMo attempts to understand why they perform the way they do by measuring nucleotide importance with saliency maps, measuring critical sequences positions for the classifier using temporal output scores, and generating class-specific motif patterns with class optimization.

Chromatin immunoprecipitation(ChIP-seq) technologies, or the precipitation of a protein antigen used to isolate and concentrate a protein from a sample containing thousands of proteins, makes finding site locations available for hundreds of different TFs. Unfortunately, ChIP-seq experiments are slow, expensive, and they can’t find patterns common across binding sites(although they can find the binding sites). Thus we cannot find why TFs bind to certain locations, and there is a need for computational methods capable of making accurate binding site classifications that can identify and understand patterns that influence the binding site locations.

Using a deep neural network, three different architectures were tested: CNN, RNN, and a combination of the two, CNN-RNN. The raw nucleotide base characters are used as input to output an output vector of a fixed dimension,

which is linearly fed to a softmax function. The final output returns a binary classification task of size 1x1 returning whether the input is a positive or negative binding site. The models use the stochastic gradient algorithm Adam with a mini-batch size of 256 sequences and a dropout regularization method.

However, deep neural networks are often criticized for their “black box” structure in that we cannot completely understand how they make predictions. DeMo attempts to understand which parts of the DNA sequence are the most influential for classification by using saliency maps. Given a sequence X_0 of length $|X_0|$ and class $c \in C$, a DNN model provides a score function of $S_c(X_0)$. Because it’s difficult to directly see the influence of each nucleotide on the complex, highly non-linear score function, given X, $S_c(X)$ can be approximated by a linear function by computing the first-order Taylor expansion(a representation of a function as an infinite sum of terms that are calculated from the values of the function’s derivatives at a single point):

$$S_c(X) \approx w^T X + b = \sum_{i=1}^{|X|} w_i x_i + b$$

For which *w* is the derivative of S_c with respect to the sequence *X* at the point X_0:

$w = \frac{\partial S_c}{\partial X}|_{X_0} =$ saliency map

The saliency map is simply a weighted sum of the input nucleotides where each weight indicates the influence of that nucleotide position on the output score.

We can find genes by looking for the start codon(ATG) and any of the three end codons. Problem is that there are 6 reading frames. Three in one direction, three in the reverse direction.

GC content is the percentage of nucleotides in a genome that are G or C. For example, a GC content of 50% would mean there is a termination codon every 64 bp. Random DNA won't show many open reading frames longer than 50 codons in length.

With bacterial genomes, codons are very easy to identify and find. Unfortunately there's more space in between real genes in eukaryotic genes(62% of human genome is intergenic), there are introns which interrupt the sequencing of DNA, codon bias(not all codons are used equally frequently), exon-intron boundaries, and many other problems. The number or frequency of codons do not necessarily dictate the complexity of an organism. *Escherichia coli* have 317 codons, humans have 450 codons, and *Saccharomyces cerevisiae* have 483 codons in their genome.

Researchers at Stanford University investigated how RNN architecture can be used to learn sequential patterns in genomic sequences, giving promising

results for epigenetics, or the study of how the genome is regulated by external factors.

Biologists in recent years have also been able to identify transcription factors(regulatory proteins that bind to a particular sequence), DNase I hypersensitive sites(sites sensitive to cleavage by the DNase I enzyme), and histone marks(chemical modifications to histone proteins). Given no additional outside information, the RNN attempts to predict whether a given feature will be present given only the sequence at the nucleotide-level.

Other models have been published in the past. DeepSEA, for example, is a deep neural network consisting of a mixture of convolutional neural networks and pooling layers which worked on a fixed length sequence, along with a sigmoid output layer to compute the probability of seeing a particular genome feature. DeepBind used a deep CNN with 4 stages of convolution, rectification, pooling, and nonlinear activation functions. Although DeepBind incorporated varying-length input, it used additional information pertaining to the features of an input sequence.

Basset, published a few months later, used a deep CNN architecture to learn the functional activity of genomics sequences.

DanQ, published in December 2015, incorporated a convolutional layer and a bidirectional long short term memory(LSTM) recurrent neural network(RNN) on top of a max pooling layer.

All of the following neural networks utilized one-hot encoding of the input sequence into the convolution layer. One-hot encoding expands integer inputs into arrays of a fixed length; for example, given that the nucleotide bases A, T, C, and G are given integer values, each letter in the sequence of DNA of length *L* would be transformed into a 4 x *L* matrix. Each tool used a convolutional layer first, which requires fixed sizes. When the length of a particular sequence cannot be discerned or is subject to change, this method becomes invalid. By introducing several sets of convolutional layers, more parameters are introduced, thus a first approach should avoid using convolutional layers.

The following loss function was used by the researchers:

$$PP(y,\hat{y}) = exp\{-\frac{1}{n-1}\sum_{t=1}^{n-1}\sum_{i=1}^{|V|} y_i^{(t)} log\hat{y}_i^{(t)}\}$$

For which *n* is the number of training samples(about the length of the genome), |V| is the size of the prediction set(4 for the 4 nucleotide bases), $\hat{y}_i^{(t)}$ is the predicted probability of the predicted word at time *t* being word *i*, and $y_i^{(t)}$ is a one-hot vector of the real word at time *t*.

For character-level processing, a bidirectional GRU is used because a genetic sequence can be regulated by both the beginning and end sequences. The outputs then go through a softmax layer; for each K task, a prediction of 0 or 1 is made which indicates whether or not a particular feature should be observed for the input sequence. The overall loss is the geometric average of

the perplexity across all K tasks. For the multitask prediction problem, |V| equals 2, the number of classes for each class.

For the character-level prediction genome task, the following datasets were prepared in order to test the ability of the neural network to detect sequences:

- A genome with a length of 30,000 where the repeated unit is AGCTTGAGGC
- A genome with a length of 30,000 random genome
- A genome with a length of 4,639,675 for *E. coli*
- A genome with a length of 23,264,338 for *P. palsifarm*(the most common parasite used to spread malaria)

The researchers used a random subset of the dataset used in the DanQ and DeepSEA papers. The dataset was collected from experiments in the ENCODE and Roadmap Epigenomics data releases. They randomly chose 80,000 sequence-label pairs for the training set, 8000 for the testing set, and 2000 for the validation set. Each sequence has a length of 1000.

At the time, training an RNN on all 1000 characters in each example proved to be particularly difficult due to computational limitations. As a result, the RNNs below were trained on the middle 100 characters of each sequence. The baseline model was trained on the same set of truncated sequences.

The first two datasets are used as testing in order to filter out any model not capable of performing well on these genomes.

The first model utilized an LSTM structure, a dropout keep rate of 1, a batch size of 50, a sequence length of 50, a learning rate of 0.002, and 10 epochs. The perplexity of the random genome returned as 4.003, meaning that the model returns the right character 1000 times out of 4003, or about 24.9% of the time. Considering that each letter(“A,” “T,” “C,” and “G”) has an equal probability of being selected in the random genome, this is a good score. The perplexity of the repeated genome was 1.002, or about 99.8% of the time the model predicted the right score. Since the sequence repeats without noise, the model should be able to put all the probability mass on a single character given the previous few characters.

For the *E. coli* and *P. falciparum* genomes, the researchers used a dropout keep rate of 1, a batch size of 50, and 1 epoch. They tested: “GRU, LSTM, and simple RNN architectures, learning rates of 0.0001 and 0.0008, sequence lengths of 50, 100, 500, and 1000, and 2 and 3 layers”(4). All combinations of the architectures returned perplexities of 3.679 for *E. coli* and 2.938 for *P. falciparum*. Although the perplexity was low, training for more than 1 epoch showed a reduction in the perplexity; however, in this experiment the number of epochs was reduced in order to test the hyperparameters and their influence on the perplexity.

Using a learning rate of 0.0008 was much faster than using a learning rate of 0.0001, although this is not surprising considering that smaller perplexities

are possible and thus increasing the learning rate in this instance would not lead to diverging from the local minima.

Using 2 or 3 layers did not significantly influence the model's ability to return smaller perplexities, and, although genomes often involve long-range interactions, sequences of 50 or 100-length sequences performed better than 500 or 1000-length sequences. RNNs and GRUs surprisingly outperformed LSTMs. On their own, the above results were not useful because genomes have certain repeated structures in certain regions, and an unequal distribution of the nucleotide bases sets a naturally low perplexity. Overall, character-level genome prediction results proved that a 2-layer network with GRUs would be most appropriate for modeling a genomic sequence and that longer input sequences do not necessarily mean lower perplexities.

The researchers also experimented with different levels of feature mapping. Mapping each of the 4 nucleotide bases to integers proved to perform quite poorly because the mapping did not encapsulate how the characters interacted with one another. Then they attempted a different type of feature mapping in which a k-mer bag of words was used, in which each version of an input sequence was counted. For k = 1, 2, 3, 4, 5, there were 1364 features. For example, in the sequence "ACTGG," feature mapping produced a length-1364 vector in which entries corresponding to the k-mers A, C, T, AC, CT, TG, GG, ACT, CTG, TGG, ACTG, CTGG, and ACTGG would equal 1; entries corresponding to G would equal two. Any other entries would equal 0.

Logistic regression was used with default parameters excluding the regularization constant of the l_2 regularization term set to 10^{-6}, and, instead of using raw error rate, the researchers looked at the F1 score defined as:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

For which precision is the proportion of correct positive predictions and recall is the proportion of positive examples that received a positive prediction. A perfect score would be 1(meaning the model has reached perfect precision and recall) and its worst at 0.

In order to account for the fact that “less than 1% of the training examples were labeled 1”(5), the baseline and RNN models only were compared when at least 1% of the training examples were labeled 1.

Ultimately, the final model used a learning rate of 0.001, 2 hidden layers, an embedding size of 2(because there were only 4 unique characters), a dropout keep probability of 0.95, hidden layer sizes of 128, a bidirectional RNN GRU with tanh activation functions, an input sequence of length 100, an Adam optimizer, and hidden layer with a size of 128.

The model, after training for 61 epochs on 80,000 training examples for 3 days on the NVIDIA GRID K520 GPU, plateaued with an average validation loss trapped at a perplexity of 1.087.

In comparison, the RNN beat the logistic regression model in 94.6% of the tasks, giving an average F1 score of 0.135 in comparison to the logistic regression model's average score of 0.044.

In conclusion, this project constructed deep bidirectional RNNs capable of predicting and capturing complex patterns in 100-character sequences of the human genome. The researchers concluded that more improvements could be made to their model:

1) Creating more data, for example, extending the 100-character sequences to 200-character sequences.
2) Modifying the hyperparameters further.
3) Designing an RNN cell suited for the long-range information associated with genomic sequences such as the clockwork RNN architecture.
4) Finding a distributed representation of genomic "words" and initializing a word-based RNN model.

Sequence prediction and classification are problems Recurrent Neural Networks(RNNs) are both designed and used for. In theory RNNs have the ability to cope with temporal dependencies; however, when long-term memory is required, they are difficult to train correctly. Using the Clockwork RNN introduces a new model in which "the hidden layer is partitions into separate modules, each processing inputs at its own temporal granularity, making computations only at its prescribed clock rate." It reduces the number of RNN

parameters by half, making it perfect for longer sequences or massive datasets and speeds up network evaluation significantly.

The researchers tested CW-RNNS on two supervised learning tasks: sequence generation where a target auto signal must be outputted without any kind of input and spoken word classification using the TIMIT dataset. Their modifications consisted of adding forward connections and partitioning neurons to the original simple RNN(SRN): “There are forward connections from the input to hidden layer, and from the hidden to output layer . . . the neurons in the hidden layer are partitioned into *g* modules of size *k*.” Each module is assigned a clock period $T_n \in \{T_1, \ldots T_g\}$ in which every module is interconnected but “the recurrent connections from module *j* to module *i* exists only if the period T_i is smaller than T_j.”

The main difference between a CW-RNN and an SRN is that at each time step *t*, only the output of modules *i* that satisfy $t \equiv 0 \,(mod\, T_i)$ are executed. At every forward pass step, only the block-rows of the hidden weight and input matrices are used for evaluation in the modulus equation, and the corresponding parts of the output vector are calculated. The low-clock-rate modules process, retain, and output the long-term information, while higher speed modules focus on high-frequency, local information.

The backward pass is the same as the SRN; however, the error propagates only from modules executed at time step *t*. The error of non-propaged modules is then passed on and copied into the back-propagated error.

Rationale

Predicting the DNA sequence is an important step in understanding many biological factors. Being able to understand the pattern in deoxyribonucleic acid allows for further technologies such as predicting and understanding transcription factor binding sites. Using architectures such as convolutional layers in neural networks, computers can accurately compile the important features of a model and, using recurrent layers such as gated recurrent units(GRUs) or long short term memory units(LSTMs), networks have the ability to retain information. Although previous sources have attempted to accurately predict the DNA sequence, restrictions on time and computational efficiency have significantly diminished research in this area. By understanding and finding patterns to genomic sequences, we can better understand how mutations such as single nucleotide polymorphisms or how DNA methylation can change phenotypes and allow scientists to design more effective treatments for genetic illnesses. Artificial RNA sequences such as shRNA can silence target gene expression; by identifying locations where certain biomarkers are such as transcription factors, histone markers, or DNase sites, illnesses with a basis in the genome have potential treatments. In addition, treatments have the ability to be specialized for different people and can reduce the risk of silencing important features. For example, in illnesses such as Crohn’s disease, a genetic basis can influence the disease and its severity; however,

there are many different factors, making the disease different for different people and making blindly silencing features potentially deadly.

Although CRISPR is a growing technology with the ability to impact millions, it is necessary to pinpoint the specific genes causing illness, and, in order to do so, an understanding of the letters being changed through CRISPR is instrumental from both an ethical and genetic standpoint.

Introduction

AI, or Artificial Intelligence, is a field of study for many. Some, such as Bill Gates and Elon Musk, believe that AI will destroy humans; however, being able to work together would be the best possible solution, as humans can interact with the physical world easily, while robots find the virtual world has favorable conditions for learning. Machine learning is a type of artificial intelligence which is capable of following instructions not explicitly given to it. Neural networks are a kind of machine learning which enable computers to predict the values of a certain situation, given inputs and their expected outputs. Their roots are derived from our own brains and how billions of neurons communicate with one another. Neural networks(NNs) are a relatively recent addition to the world of artificial intelligence which has taken the world by storm. In biology, NNs have proven incredibly useful as they are capable of tasks such as identifying skin lesions or classifying genomic sequences. Their ability to make complex networks out of a simple perceptron replicates the processes occurring in our brain as axioms interact with one another and allow us to make conclusions and decipher problems. This project uses neural networks in order to identify 919 chromatin features within a genomic sequence: 125 DNase features, 690 transcription factor binding sites, and 104 histone predictions. There are more than 10,000 people who suffer from monogenic disorders, or disorders involving a single error in the genetic code. Viruses kill. Diseases are

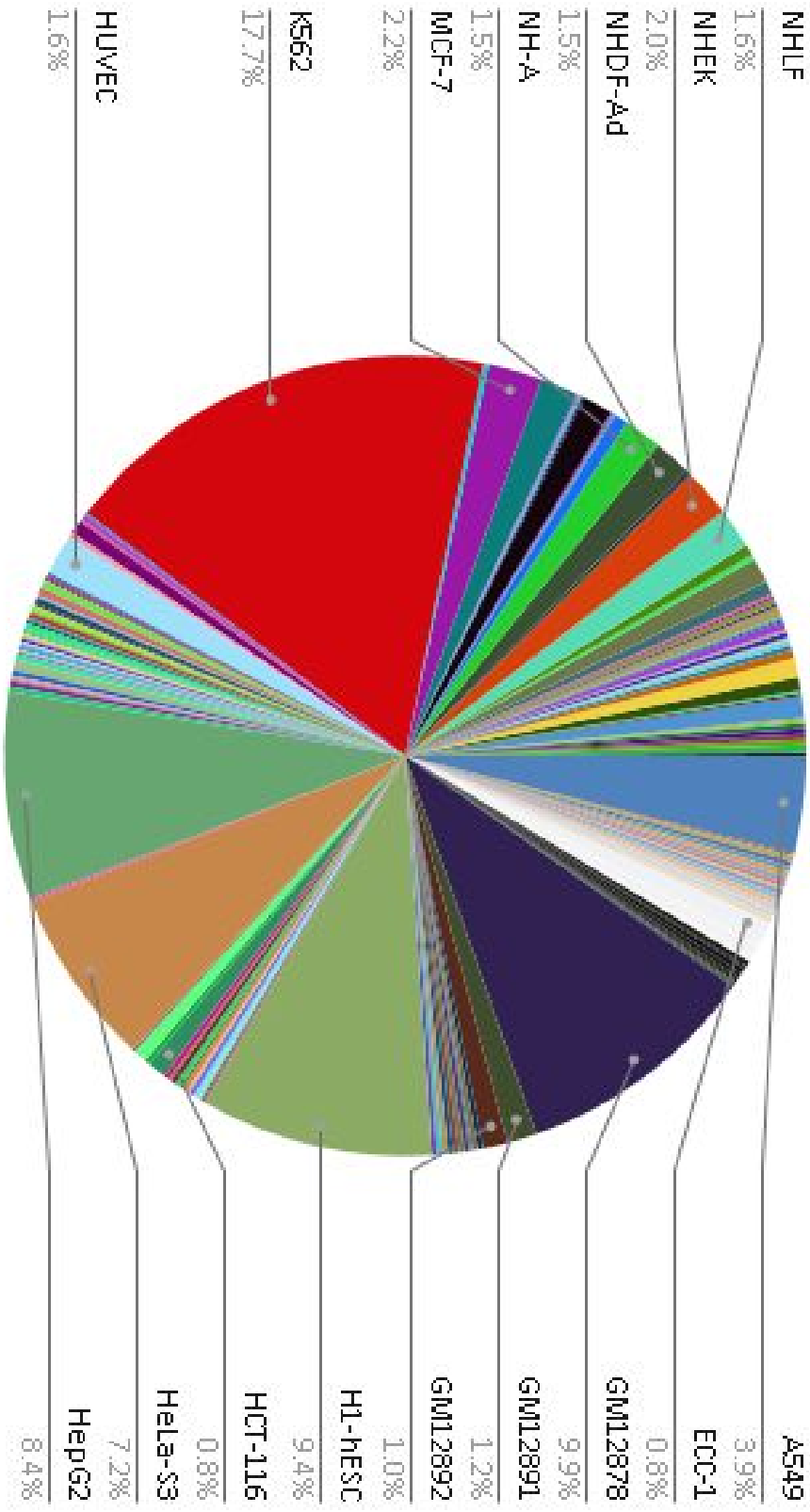
unstoppable in their relentless pursuit to evolve and destroy our immune system over and over again. Yet there is hope. Using machine learning, we can double our efforts in finding the cure to illnesses while neural networks can aid in the tedious task of finding and identifying chromatin features in the genome.

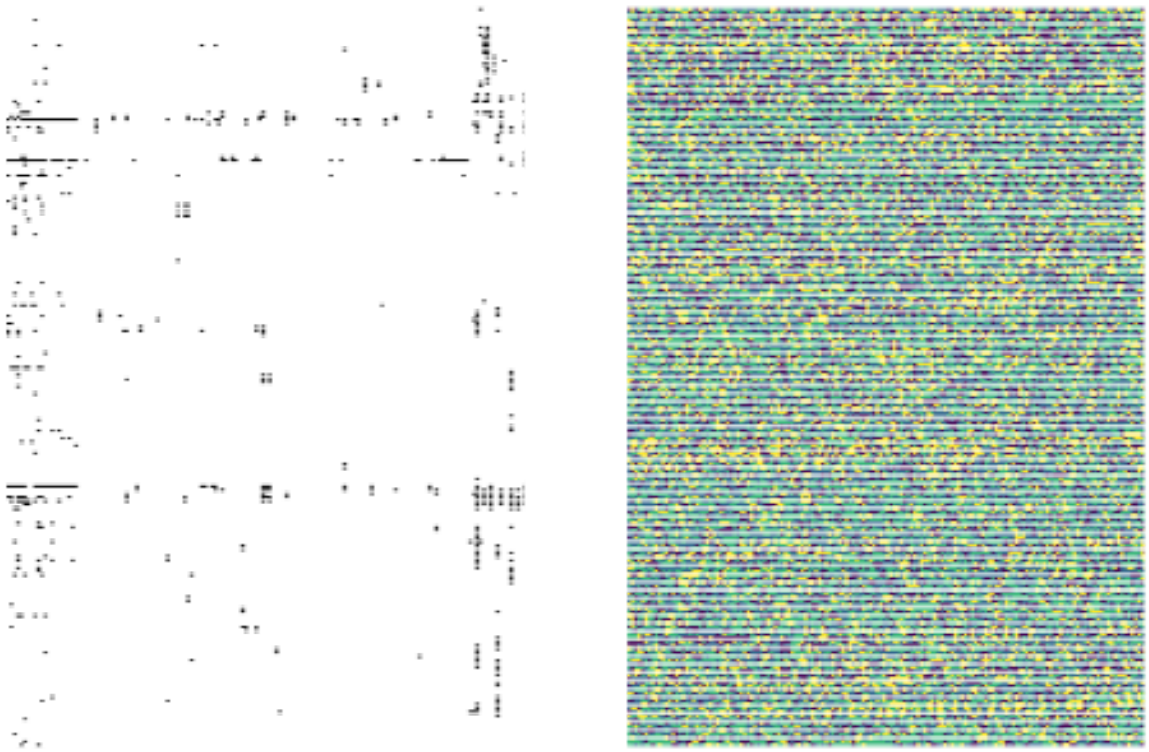
Purpose

Is it possible to predict chromatin features in a genomic sequence, and, if so, how can this technology aid in diagnosing illnesses?

Hypothesis

The researcher hypothesizes that, by extracting features such as transcription factors in a genomic sequence in coordination with deep neural network methods including gated recurrent units and clockwork RNNs, genomic sequence prediction and classification is possible and understandable using features described in the Deep Motif Dashboard(DeMo), created by researchers at the University of Virginia. Using the first-order Taylor expansion, a representation of the model as an infinite sum of terms calculated from the values of the model's derivatives at a single point, a saliency map can be derived in order to determine the influence of a nucleotide position on the output score. Combining two works focusing separately on optimizing different elements of genomic sequence prediction and comprehension should aid in creating a complex model capable of extracting features accurately.

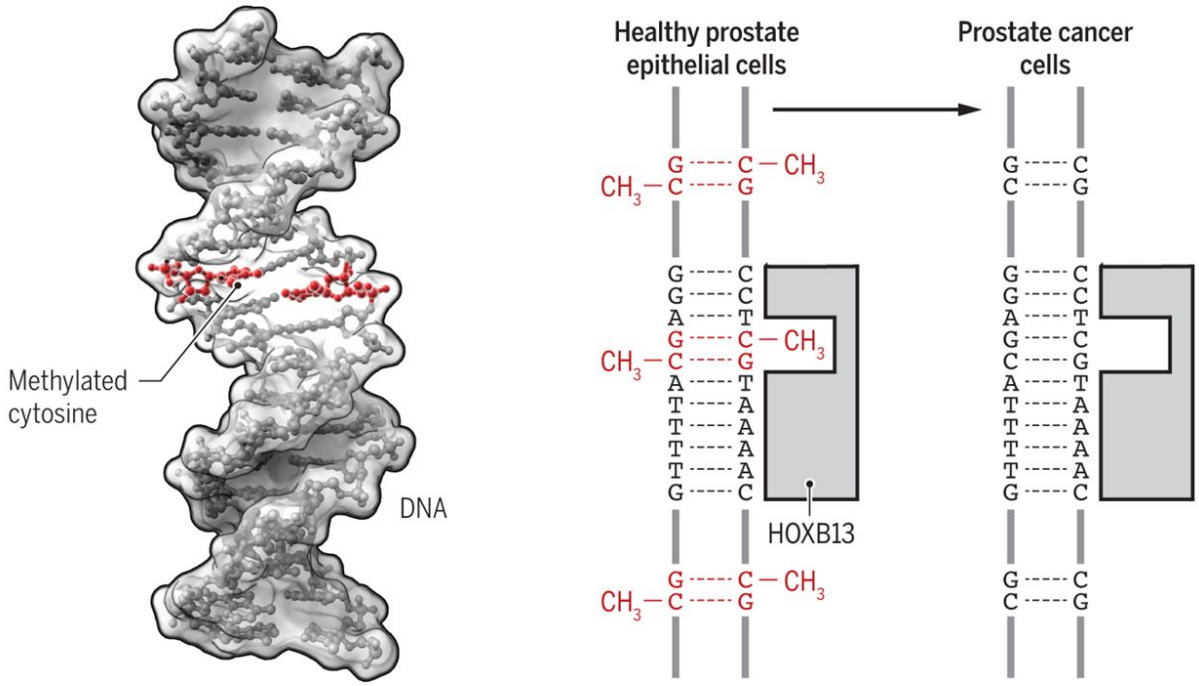

Count of cell line
NHLF
1.6%
NHEK
2.0%
NHDF-Ad
1.5%
NH-A
1.5%
MCF-7
2.2%
K562
17.7%
HUVEC
1.6%
A549
3.9%
ECC-1
0.8%
GM12878
9.9%
GM12891
1.2%
GM12892
1.0%
H1-hESC
9.4%
HCT-116
0.8%
HeLa-S3
7.2%
HepG2
8.4%



Sample inputs and outputs for the NN - the right is a feature map of the first 200 outputs. Each black dot corresponds to a chromatin feature present. To the left is a feature map of the first 200 inputs for which each letter corresponds to a different nucleotide base - yellow representing A, green representing T, blue representing C, and purple representing G.

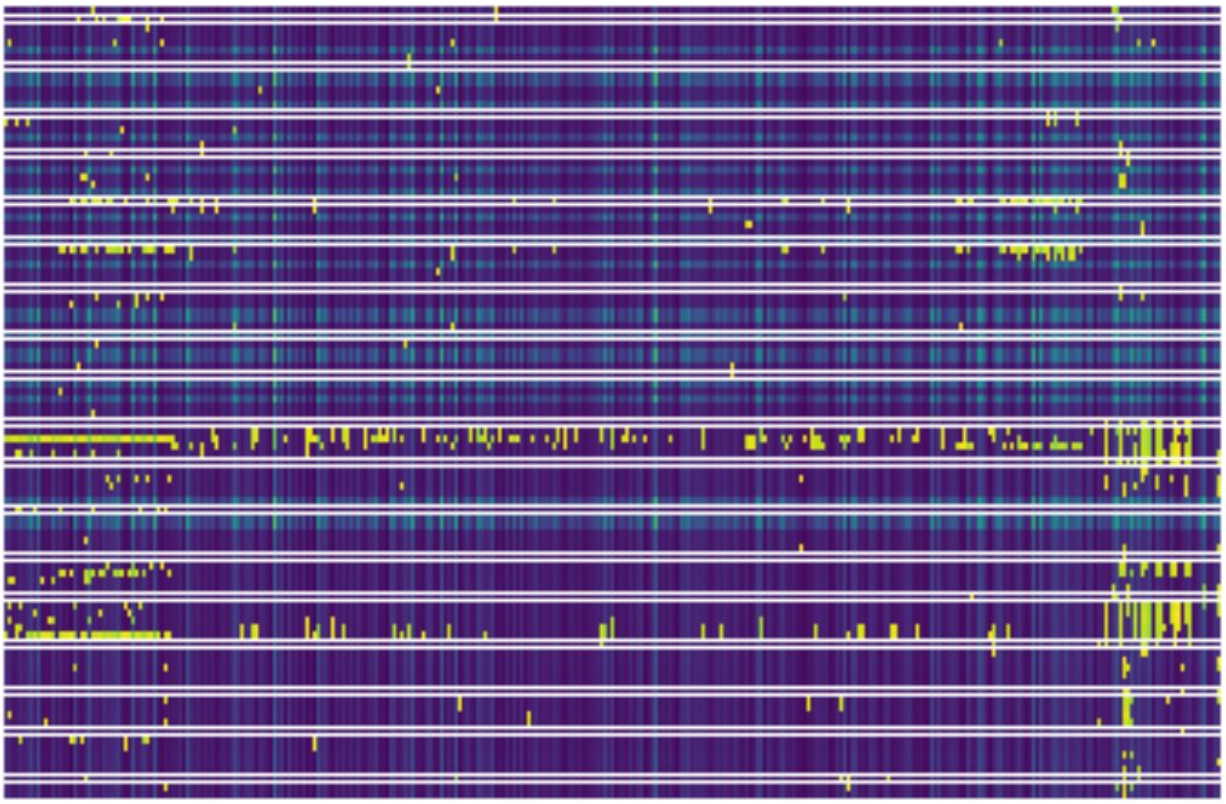
New binding sites

A methylated DNA structure (PDB:5EF6) is shown. "Methyl-plus" binding may be a pioneering mechanism for transcription factors such as HOXB13, which can also bind to unmethylated DNA in cancer cells.

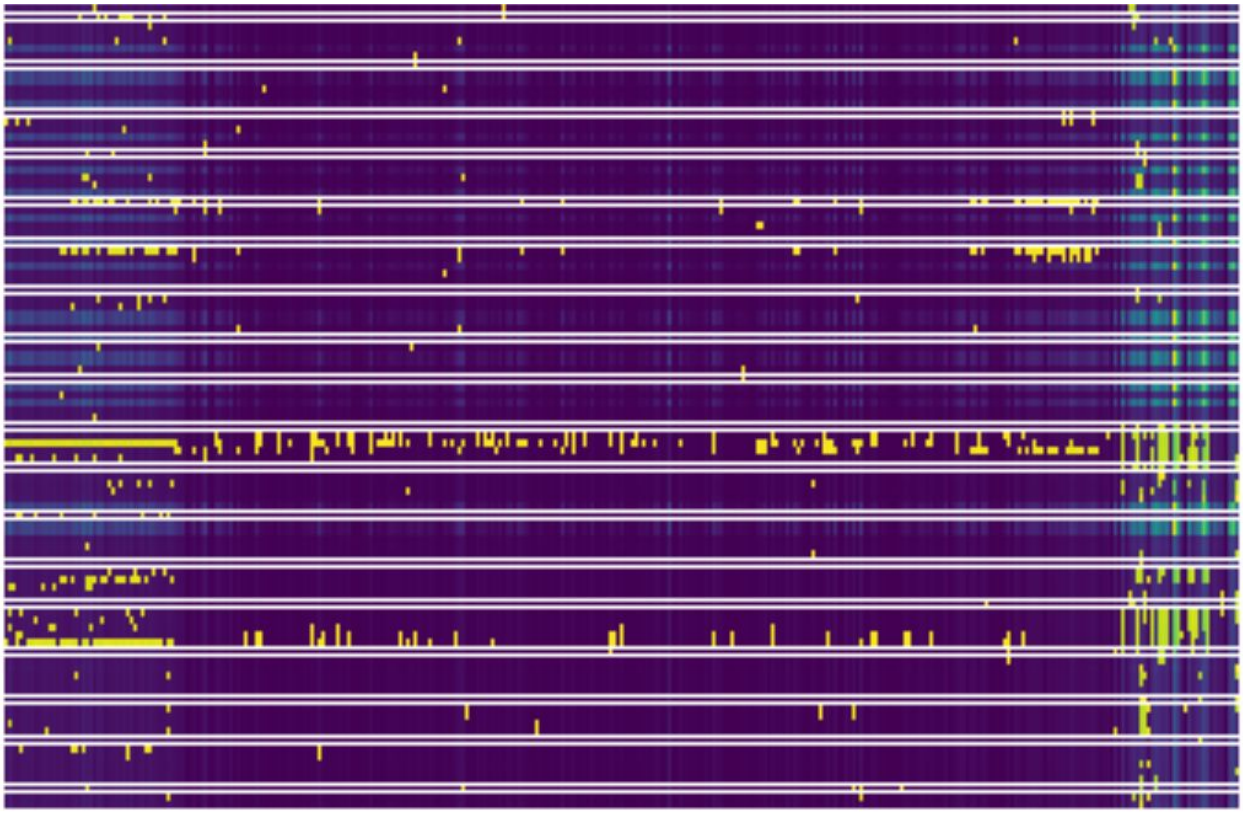


Source: https://science.sciencemag.org/content/sci/356/6337/489/F1.large.jpg

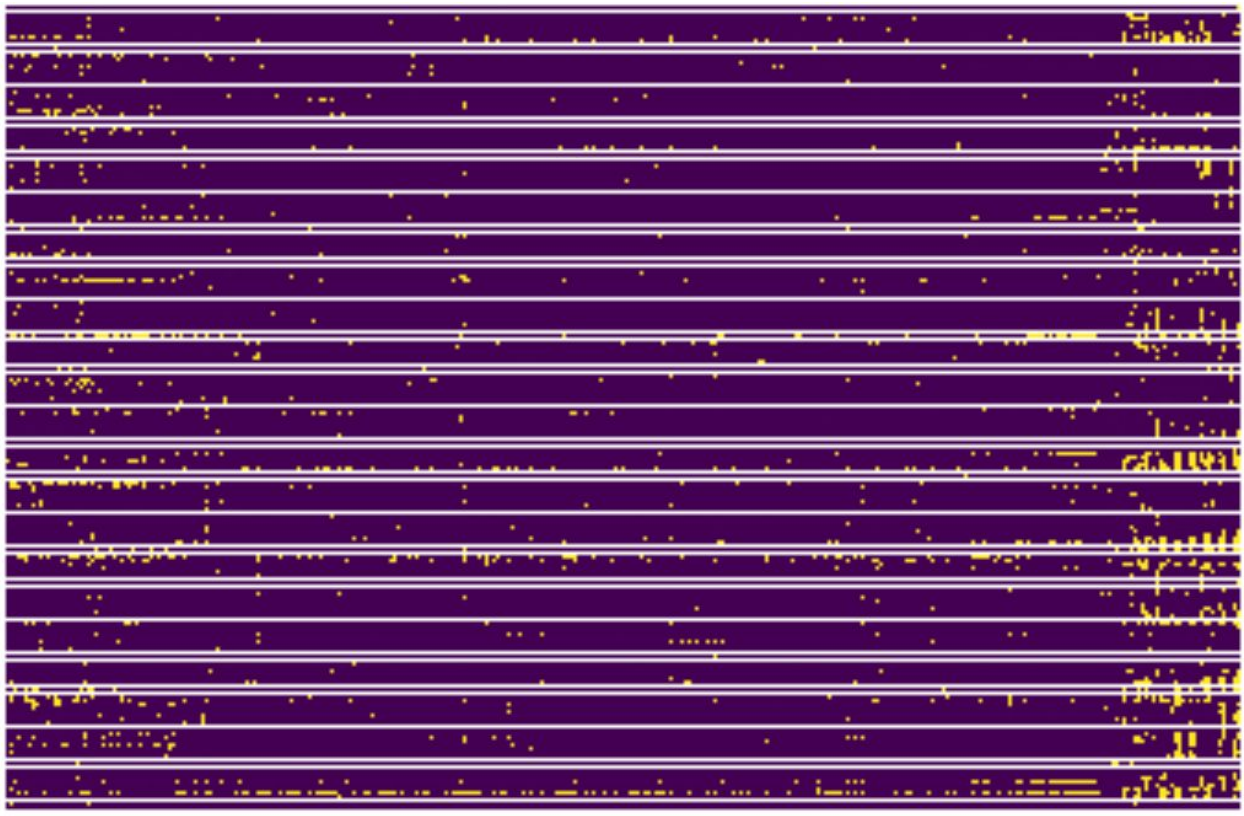
Batch 1, Epoch 1:



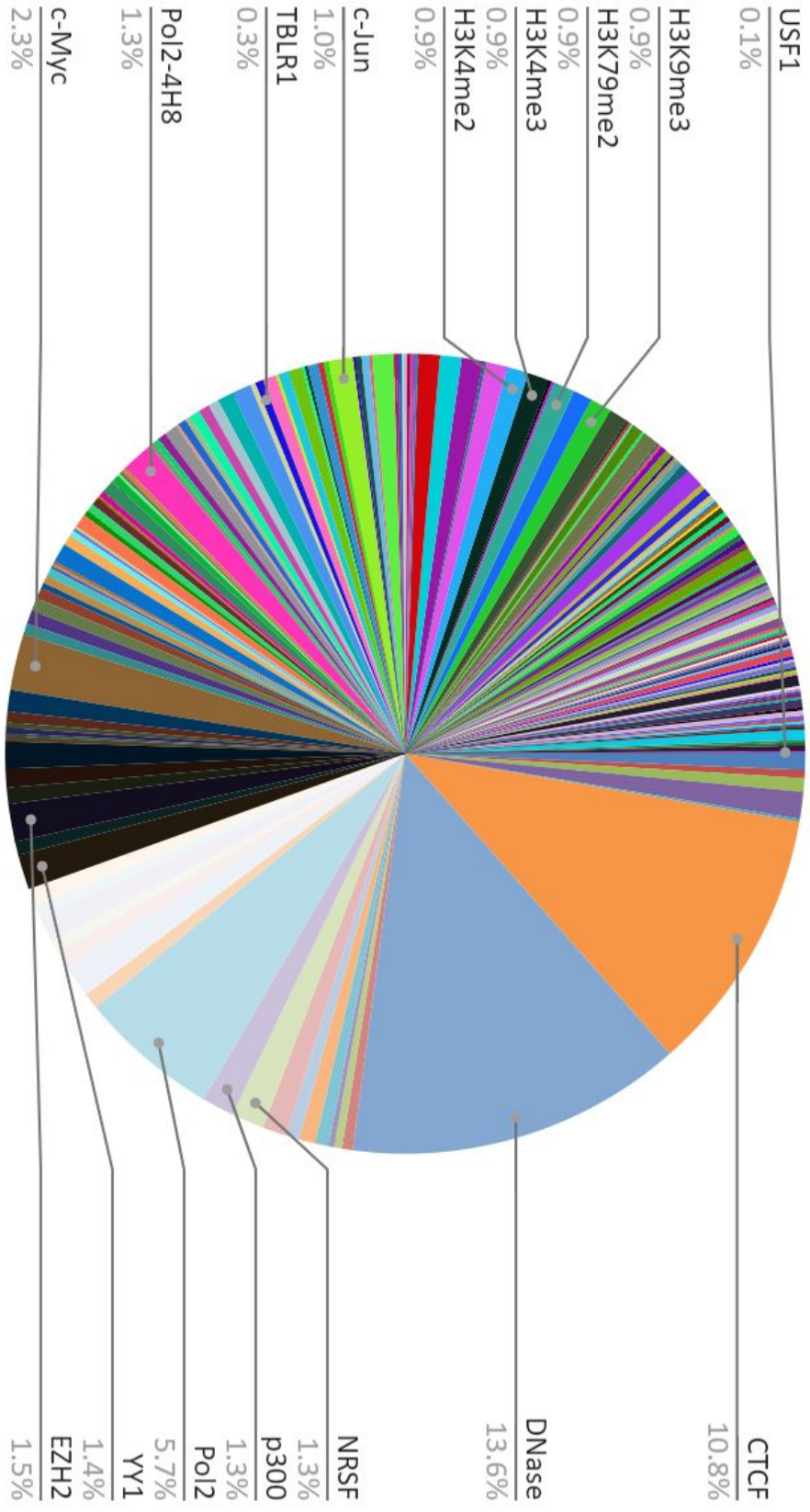
Batch 1, Epoch 10:

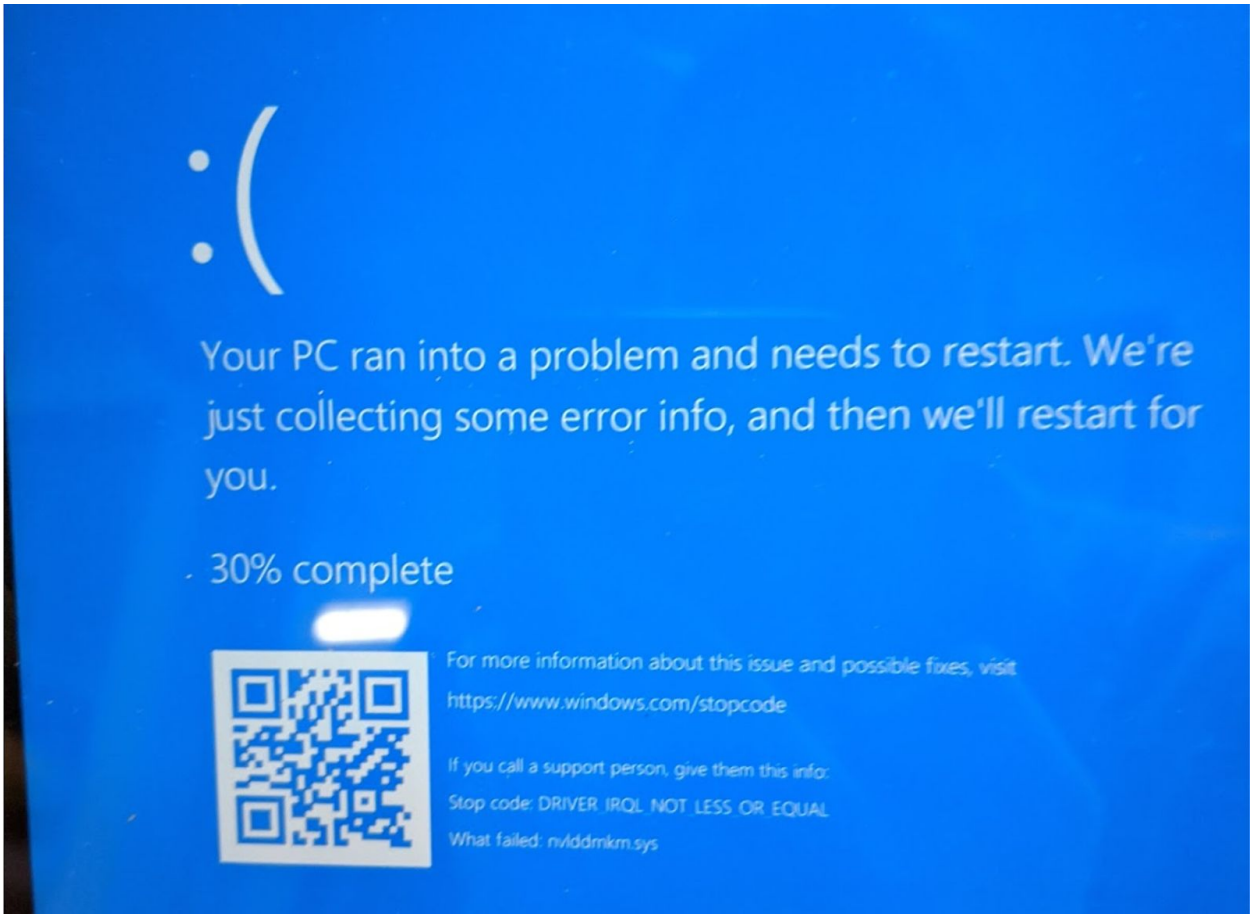


A feature map of the differences between the output and the predicted output of the neural network.



This feature map compares the differences between the validation output and the training output.


Count of Regulatory Elements
USF1
0.1%
H3K9me3
0.9%
H3K79me2
0.9%
H3K4me3
0.9%
H3K4me2
0.9%
c-Jun
1.0%
TBLR1
0.3%
Pol2-4H8
1.3%
c-Myc
2.3%
CTCF
10.8%
DNase
13.6%
NRSF
1.3%
p300
1.3%
Pol2
5.7%
YY1
1.4%
EZH2
1.5%



The “blue screen of death” encountered in response to a memory overload.

The Code

This code works given train.mat, a file taken from DeepSea's database. The Python libraries TensorFlow, h5py, os, NumPy, and Matplotlib must be installed beforehand.

transcriptionTesting.py

```
# Imports
import tensorflow as tf
print(tf.__version__)
import h5py, os
import numpy as np
import matplotlib.pyplot as plt
fileI = "train.mat" # In some cases(eg, VSC running .ipynb file) this path may need to be defined statically.
fileVal = "valid.mat"
print("Everything is more or less working.")
# Loading data
# This cell uses a LOT of RAM and battery power.
# train.mat contains 3.4GB of data, but, in some cases,
# RAM usage is much higher. This could be caused by:
"""
- Other processes
- The source running the .ipynb file(eg VSC has many extra features == extra RAM usage)
- Multiple variables storing the file; shallow copying may account for this, but it still remains a problem.
"""

print(os.path.exists(fileI))
f = h5py.File(fileI, "r+")
print(f.keys())
inputs = f.get("trainxdata")[()]
print(inputs.shape)
print(inputs[0])
outputs = f.get("traindata")[()]
```

```
print("Loaded train data.")
# Outputs are 919 chromatin features.
# 125 DNase features, 690 TF features, and 104 histone predictions.
# Our inputs, on the other hand,
# Are of length (1000, 4)
# Deleting the initial training array(f) aids in decreasing memory usage a little.
del f
print(inputs[0])
threshold = int(len(inputs) * 3/4)
inputVal = np.transpose(inputs, (2, 0, 1))[threshold:]
outputVal = outputs[:,threshold:].T
print(inputVal.shape, outputVal.shape)
inputs = np.transpose(inputs, (2, 0, 1))[:threshold]
outputs = outputs[:,:threshold].T
print(inputs.shape, outputs.shape)
# Here is some data to show
# This is the input; we can only show a certain
# Amount of letters before it doesn't look nice anymore,
# So here is a barcode-like representation:

# plt.imshow(inputs[8][:100])
# plt.show()
# Here is the output(validation set), displayed as a bar code.
def argmaxFunc(a):
    return np.array([np.argmax(i) for i in a])
# argmaxFunc = np.vectorize(myFunc)
def draw(inputs, outputs, max_len=200):
    subplots = plt.subplots(200, 2)
    for i in range(len(subplots[1])):
        subplots[1][i][0].set_axis_off()
        subplots[1][i][0].imshow(outputs[i].reshape((1, -1)), aspect="auto",
cmap="binary", interpolation=None)
        subplots[1][i][1].set_axis_off()
        subplots[1][i][1].imshow(argmaxFunc(inputs[i]).reshape((1, -1)), aspect="auto",
interpolation=None)
    return plt
draw(inputVal, outputVal).show()
```

```
# Looking at the data
infoFile = "journal.pcbi.1007616.s007.xlsx"
import xlrd
fullCellData = []
workbook = xlrd.open_workbook(infoFile)
worksheet = workbook.sheet_by_index(0)
for row in range(1, worksheet.nrows):
    fullCellData.append({
        "cell": worksheet.cell_value(row,0),
        "regulatory element": worksheet.cell_value(row,1),
        "treatment": worksheet.cell_value(row, 2)
    })
# Alright, we're going to
# - Create a function which takes in the output array,
# - Puts it in alignment with all the other info we have,
# - And graphs the normalized probability of that occurring.
# fullCellData = np.array(fullCellData)
def visualizeOutput(output):
    # Taken in output, let's try a histogram.
    val = 0 # Where the data will appear
    # We can plot the data as a simple scatter plot, as shown here.
    plt.plot(output, color="green", marker="o", markersize=2)
    plt.show()
    # We can transform it into a pie chart, which will take a little more work....
    chart = fullCellData
    # Now we can show a chart
    for i in range(len(chart)):
        chart[i]["output"] = output[i]
    chart.sort(key=lambda val: val["output"], reverse=True)
    # print(chart[:10])
    outputPush = []
    for i in range(len(chart)):
        if i % 100 == 0:
            print("%d percent through" % int(i / len(chart)*100))
        outputPush.append(list(chart[i].values()))
    plt.table(cellText=outputPush,
```

```
    loc="center",
    colLabels=["Cell Lines", "Regulatory Element", "Treatment", "Probability"])
    plt.savefig("PDF.pdf")
    plt.show()
# np.savetxt("info.csv", outputVal[0], delimiter=",")
visualizeOutput(outputVal[0])

# # Now there is some more file assorting.
# # We have a file with the names of all the chromatin features we're looking for;
problem is, they are in files.txt.
# fileO = []
# timesBefore = 1
# for i in open("files.txt", "r+").read().split("\n"):
#     if len(i.split("; ")) >= 6:
#         if len(fileO) > 0:
#             cellName = i.split("; ")[6][5:]
#             if cellName != fileO[-1]:
#                 fileO.append(cellName)
#             else:
#                 print("Skipped " + str(timesBefore))
#                 timesBefore += 1
#         else:
#             fileO.append(i.split("; ")[6][5:])
#     else:
#         print("SKIPPED")
# print(len(fileO), fileO)
# Testing whether we have a GPU or not
# As of TF 2, GPU support is used by default, so this only applies
# If we have TF version < 2.
if int(tf.__version__[0]) < 2:
    if tf.test.is_gpu_available():
        rnn = tf.keras.layers.CuDNNGRU # This checks if it can use CuDNNGRU.
        print("GPU support enabled.")
    else:
        import functools
        rnn = functools.partial(
          tf.keras.layers.GRU, recurrent_activation='tanh')
```

```
        print("GPU not found, defaulting to CPU.")
else:
    rnn = tf.keras.layers.GRU
    if tf.test.is_gpu_available():
        print("GPU support enabled.")
    else:
        print("GPU will NOT be used. Make sure Cuda is in your PATH.")

# F1 metric
def f1_metric(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
# Now we're going to work on the actual model.
# --------TEST MODEL----------
def Model(input_shape, output_shape, unit1=128, unit2=128):
    inputs = tf.keras.Input(input_shape[1:])
    # Simple RNN
    recurrent = rnn(unit1, return_sequences=True)(inputs)
    recurrent1 = rnn(unit2, return_sequences=False)(recurrent)
    dense1 = tf.keras.layers.Dense(output_shape[1], activation="sigmoid")(recurrent1) #
return_state
    # Dense layer
    # dense = tf.keras.layers.Dense(919)(recurrentLayer) #919
    print(dense1.shape, inputs.shape)
    model = tf.keras.Model(inputs=inputs, outputs=dense1)
    return model

# This very simple model is merely a proof of purpose.
# Just to see whether the shapes work, whether the GRU
# Performs more or less correctly, etc.
# Actual training
# Creating the model
```

```
print(tf.__version__)
import tensorflow.keras.backend as K
print(inputs.shape, outputs.shape)
print(len(inputs[0][0]))
model = Model(inputs.shape, outputs.shape)
# We will need to reverse the shape in order for this to work
# Work properly, but first let's check whether this works.
accReadings = []
f1Readings = []
perplexity = []
class MyCallback(tf.keras.callbacks.Callback):
    def on_train_batch_end(self, batch, logs=None):
        # print(logs["loss"], tf.math.exp(logs["loss"]))
        perplexity.append(np.exp(logs["loss"]))
        accReadings.append(logs["loss"])
        print(np.exp(logs["loss"]))
        print("Batch ended with a perplexity of %2d" % np.exp(logs["loss"]))
    def on_epoch_end(self, batch, logs=None):
        draw(inputVal, model.predict(inputVal)).show()
model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy", f1_metric])
model.summary()
#
# Now that the model is created, we finally
# Can try out the training.
# Proper model
# Taken from DeepSea model
def Model(input_shape, output_shape):
    # Input
    model = tf.keras.Sequential()
    model.add(tf.keras.Input(input_shape[1:]))
    """
    "The basic layer types in out model are convolution layer, pooling layer and fully
connected layer. A convolution layer computes output by one-dimensional convolution
operation with a specified number of kernels . . . . In the first convolution layer, each
```

```
kernel can be considered as a position weight matrix(PWM) and the convolution operation
is equivilent to computing the PWM scores with a moving window with step size one on the
sequence."
    """
    """Here is the code in Lua:

model:add(nn.SpatialConvolutionMM(nfeats, nkernels[1], 1, 8, 1, 1, 0):cuda())
model:add(nn.Threshold(0, 1e-6):cuda())
model:add(nn.SpatialMaxPooling(1,4,1,4):cuda())
model:add(nn.Dropout(0.2):cuda())

model:add(nn.SpatialConvolutionMM(nkernels[1], nkernels[2], 1, 8, 1, 1, 0):cuda())
model:add(nn.Threshold(0, 1e-6):cuda())
model:add(nn.SpatialMaxPooling(1,4,1,4):cuda())
model:add(nn.Dropout(0.2):cuda())

model:add(nn.SpatialConvolutionMM(nkernels[2], nkernels[3], 1, 8, 1, 1, 0):cuda())
model:add(nn.Threshold(0, 1e-6):cuda())
model:add(nn.Dropout(0.5):cuda())

nchannel = math.floor((math.floor((width-7)/4.0)-7)/4.0)-7
model:add(nn.Reshape(nkernels[3]*nchannel))
model:add(nn.Linear(nkernels[3]*nchannel, noutputs))
model:add(nn.Threshold(0, 1e-6):cuda())
model:add(nn.Linear(noutputs , noutputs):cuda())
model:add(nn.Sigmoid():cuda())
"""
    nkernels = [4, 320,480,960]
    dropout = [0.2, 0.2, 0.5]
    # We have 3 rounds of convolution. Each one contains a convolution layer with output
of kernel size nkernels[i], a threshold(which we can implement later...), and a dropout.
    model.add(tf.keras.layers.Conv1D(4, 2, 1, "valid"))
    model.add(tf.keras.layers.MaxPooling1D(2))
    model.add(tf.keras.layers.Dropout(dropout[i]))
    model.summary()
    return model
print(inputs.shape, outputs.shape)
```

```
model = Model(inputs.shape, outputs.shape)
```

Procedure

1) Obtain and download the data from:

- The Deep Motif Dashboard(https://media.githubusercontent.com/media/QData/DeepMotif/master/data/deepbind.tar.gz)

- The ENCODE database(http://deepsea.princeton.edu/media/code/deepsea_train_bundle.v0.9.tar.gz)

- The *E. coli* genome(ftp://ftp.ensemblgenomes.org/pub/bacteria/release-46/fasta/bacteria_0_collection/escherichia_coli_str_k_12_substr_mg1655/dna/)

2) Download Pip, Python 3.7.x, and the following Python libraries:

- timeit

- TensorFlow

- OS

- h5py

- NumPy

- Keras

- Matplotlib

3) Begin programming, using Python as a mainframe.

a) Work on a character-level prediction neural network. This neural network should take in N number of input letters and output N output letters, using the k-mer bag of words method.

b) Work on a binary classification prediction neural network, taking in N number of input letters and outputting whether there are clear transcription factors in the DNA.

c) Work on modification of the neural network by implementing graphs using the Matplotlib library.

d) Work on efficiency of the neural network by comparing the speed of the GPU and CPU, running a simple test(by timing how long it takes each processing unit to run a convolutional 32x7x7x3 filter over random 100x100x100x3 vectors) and utilizing whichever works fastest. Although typically the GPU works significantly faster, in some cases and in past bugs TensorFlow has been revealed to have efficiency problems with GPUs not configured correctly or certain older GPUs.

Materials

- 1 computer(the researcher is using an HP Envy 13 with an NVIDIA GeForce MX250 GPU)

Conclusion

Ultimately this project resulted in creating a neural network capable of understanding genomic sequences and predicting 919 chromatin features: 125 DNase features, 670 transcription factor features, and 104 histone features. Using data scraped from the ENCODE and Roadmap epigenomics databases, the final product depended heavily on DeepSea by building on their dataset and working on incorporating different techniques used in other studies such as recurrent neural networks and gated recurrent units(GRUs). This project also focused particularly on comprehension and analysis of the data by using the "Matplotlib" library in order to render the inputs and outputs as one dimensional feature maps and aligning the different chromatin features with their corresponding cell types, regulatory elements, and treatments.

Problems Encountered

The researcher encountered a multitude of problems while working on this project. DeepSea utilized a desktop PC with massive computing power while the researcher used a smaller PC with a relatively high-end GPU not only because of the expense but also as a proof of purpose; neural networks can be written on low-end PCs and perform similarly because ultimately, even with a lower-end processor, NNs can still grasp the concepts and prove that patterns do exist in data. Previously the researcher used .py files; however, they were found to be time-expensive due to the massive size of the training file and the time it took to load which significantly slowed advancements in writing the code. The researcher ultimately used .ipynb files, a type of Python file which allows for running pieces at one time. Different Python environments contributed to errors as different Tensorflow environments allowed for different capabilities and restrictions. The newest version of Tensorflow, 2.1, allows for Tensorflow add ons such as a built-in F1 metric; however, tf.contrib, referenced multiple times in the original code, was removed from that version of Tensorflow, meaning that either the researcher could enable the F1 metric and scrap the tf.contrib components or disable the F1 metric altogether and keep the original code. Ultimately the researcher incorporated the F1 metric by hand and downgraded Tensorflow to a more usable version less focused on detail.

The researcher also found that past research in this area has proven to be very general and not explained in sufficient detail to be potentially replicated by other researchers. Dr. Jessez Zhang, for example, provided an accurate description of his experiments with enough parameters defined to be replicated. DeepSea, on the other hand, provided specific code; however, the code was most likely not the final version used because they designed a convolutional neural network with invalid parameters. Not only were the filters set to obscene values, the CNN was set to take in a three dimensional input when the research paper explicitly stated that the CNN would take in a two dimensional input and the reshaping of the input in the code corresponded to reshaping the data to a two dimensional input.

Future Expansions

Potentially this project, given more time and more processing power, could attempt to use more parameters, or perceptrons; because the memory limit was already very close to capacity, the level of complexity the model could have was very low. Potentially this project could also aid in designing antibodies to combat viruses such as the coronavirus.

Practical Applications

This project could be applied in modern medicine in order to use technologies such as CRISPR or shRNA to silence features such as transcription factors. For example, the treatment for the coronavirus lies in targeting biomarkers and silencing the part that allows them to attack our immune systems and render us useless. This also could provide the cure to rare illnesses in which very little time has been spent determining the factors which influence these diseases such as Crohn’s disease.

Bibliography

Brown, Terence A. “Understanding a Genome Sequence.” Genomes. 2nd Edition., U.S. National Library of Medicine, 1 Jan. 1970, www.ncbi.nlm.nih.gov/books/NBK21136/.

Culex Pipiens(Diptera: Culicidae) to Transmit West Nile Virus.” Journal of Medical Entomology, vol. 39, no. 1, Jan. 2002, pp. 221–225., doi:10.1603/0022-2585-39.1.221.

“Escherichia Coli Str. K-12 Substr. MG1655.” Escherichia Coli Str. K-12 Substr. MG1655 - Ensembl Genomes 46, bacteria.ensembl.org/Escherichia_coli_str_k_12_substr_mg1655/Info/Index.

Koutník, Jan. “A Clockwork RNN.” Cornell University, 14 Feb. 2014, arxiv.org/abs/1402.3511.

Lanchantin, Jack, et al. “Deep Motif Dashboard: Visualizing and Understanding Genomic Sequences Using Deep Neural Networks.” Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, U.S. National Library of Medicine, 2017, www.ncbi.nlm.nih.gov/pmc/articles/PMC5787355/.

Redell, Michele S, and David J Tweardy. “Targeting Transcription Factors in Cancer: Challenges and Evolving Strategies.” ScienceDirect, Elsevier, 31 Oct. 2006, www.sciencedirect.com/science/article/abs/pii/S1740674906000588.

Zhang, Jesse M, and Govinda M Kamath. “Learning the Language of the Genome Using RNNs.” Deep Learning for Natural Language Processing, cs224d.stanford.edu/reports/jessesz.pdf.

Zhou, Jian, and Olga G Troyanskaya. “Predicting Effects of Noncoding Variants with Deep Learning-Based Sequence Model.” *Nature Methods*, U.S. National Library of Medicine, Oct. 2015, www.ncbi.nlm.nih.gov/pmc/articles/PMC4768299/.